Unit –5: Polymorphism, Virtual Function and Working with Files

This Pointer

- The this pointer holds the address of current object. In other words we can say that this pointer points to the current object of the class.
- This is a pointer to object.
- This pointer is used to represent an object that invokes a member function.
- **Example:** the function call A.max( ) will set the pointer this to the address of the object A.
- Friend functions do not have a this pointer, because friends are not members of a class. Only member functions have a this pointer.
- Static member functions don't have a this pointer.

1) The private variable can be directly inside a member function

```
class ABC
{
        int a;
        public:
        void getdata(int a)
        {
                This ->a=a;
        }
        void print()
        {
        cout << "a = " <<a << endl;
        }
};
```

```
 void main( )
{
        ABC a1;
        a1.getdata(50);
        a1.print( );
}
```

Output:

a=50

```
class Test
{
        private:
        int x;
        int y;
        public:
        Test(int x = 0, int y = 0)
        {
                this->x = x;
                this->y = y;
         }
        Test &setX(int a)
        {
                 x = a;
                return *this;
        }
```

```
Test &setY(int b)
        {
                 y = b;
                return *this;
        }
void print()
{
         cout <<x << y << endl;
} };
int main()
{
  Test obj1(5, 5);
  obj1.print();
  obj1.setX(10).setY(20);
  obj1.print();
  return 0;
}
```

2) The application of this pointer is to return the object it points to.

```
class person
{
        private:
        char name[10];
        int age;
        public:
        void getdata( )
        {
        cout<<”Enter Name and Age :”
        cin>>Name;
        cin>>age;
        }
void putdata( )
{
        cout<<”Name:”<<name;
        cout<<”Age:”<<age;
}
```

```
person & elder (person p1, person p2)
{
        if (p2.age > p1.age)
        return p2;
        else
        return *this;
}
};
void main( )
{
        person p1,p2;
        p1.getdata( );
        p2.getdata( );
        p1.putdata( );
        p2.putdata
```

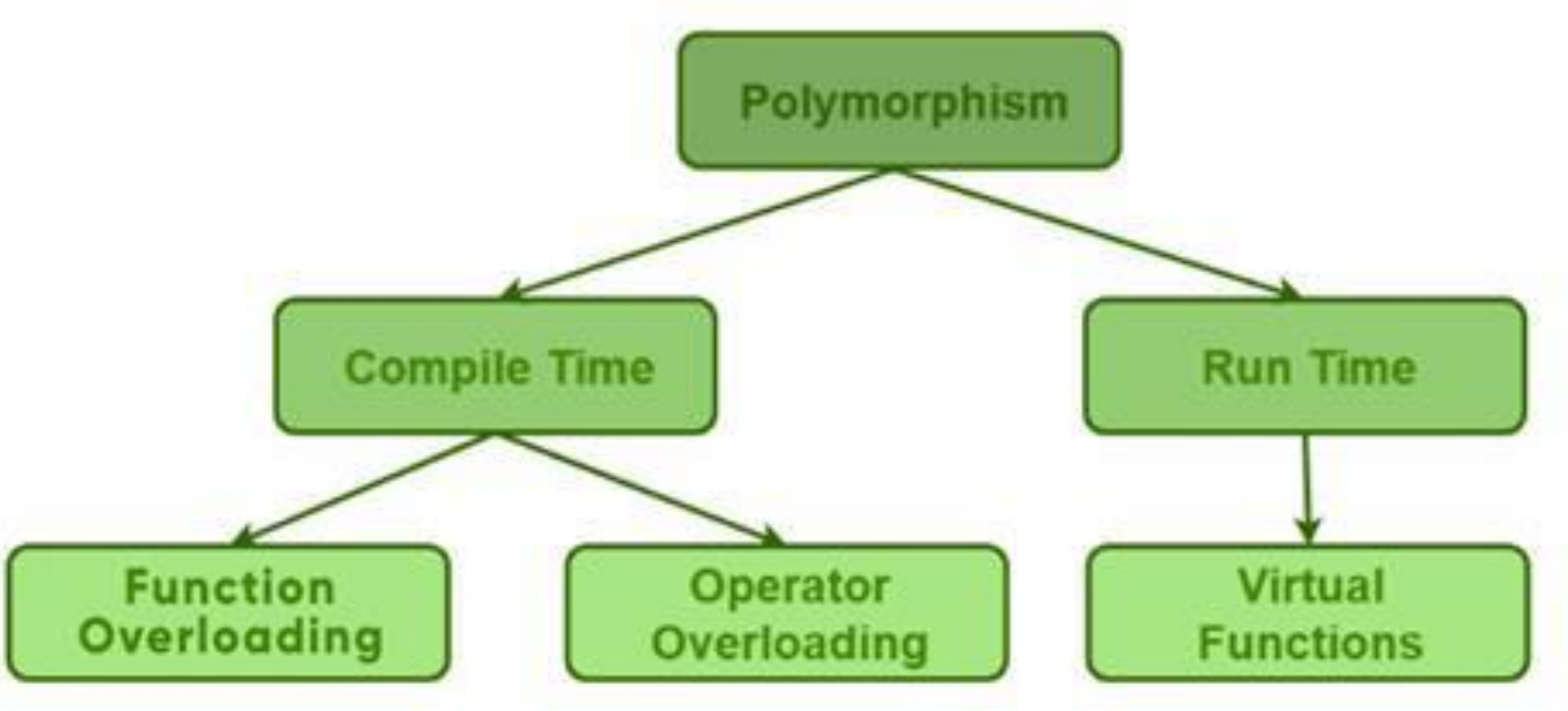
```
        person& p3=p1.elder(p1,p2);
        cout<<" Elder is : ";
        p3.putdata( );
        getch( );
}
```

Output:

```
Enter Name and Age : Sunita 20
Name: Sunita
Age: 20
Enter Name and Age : Anita 18
Name: Anita
Age: 18
Elder is :
Name: Sunita
Age: 20
```

Compile time and runtime polymorphism

- Polymorphism means "many forms".
- In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.



Function Overloading in C++

- Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters.
- When a function name is overloaded with different jobs it is called Function Overloading.
- In Function Overloading “Function” name should be the same and the arguments should be different.
- Function overloading can be considered as an example of a polymorphism feature in C++.

```
#include <iostream>
using namespace std;
void add(int a, int b)
{
  cout << "sum = " << (a + b);
}
void add(double a, double b)
{
    cout << endl << "sum = " << (a + b);
}
```

```
int main()
{
    add(10, 2);
    add(5.3, 6.2);
    return 0;
}
```

Output

```
sum = 12
sum = 11.5
```

Operator Overloading in C++

- In C++, Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.
- C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. Operator overloading is a compile-time polymorphism.

Example:

```
int a;
float b,sum;
sum = a + b;
```

- Here, variables “a” and “b” are of types “int” and “float”, which are built-in data types.
- Hence the addition operator ‘+’ can easily add the contents of “a” and “b”.
- This is because the addition operator “+” is predefined to add variables of built-in data type only.

```
class A {
        statements;
};
int main()
{
        A a1, a2, a3;
        a3 = a1 + a2;
}
```

Now, if the user wants to make the operator “+” add two class objects, the user has to redefine the meaning of the “+” operator such that it adds two class objects. This is done by using the concept of “Operator overloading”.

Syntax for C++ Operator Overloading

- The syntax for overloading an operator is similar to that of function with the addition of the operator keyword followed by the operator symbol.

```
returnType operator symbol (arguments)
{        ... .. ...
}
```

returnType - the return type of the function

operator - a special keyword

symbol - the operator we want to overload (+, <, -, ++, etc.)

arguments - the arguments passed to the function

Difference between Operator Functions and Normal Functions

- Operator functions are the same as normal functions.
- The only differences are, that the name of an operator function is always the operator keyword followed by the symbol of the operator, and operator functions are called when the corresponding operator is used.

Overloading the Binary + Operator

```
class Complex
{
private:
int real, imag;
public:
Complex(int r = 0, int i = 0)
{
    real = r;
    imag = i;
}
Complex operator +(Complex const& obj)
{
      Complex res;
      res.real = real + obj.real;
      res.imag = imag + obj.imag;
      return res;
}
```

```
void print()
{
        cout << real << " + i" << imag << '\n';
}
};
int main()
{
   Complex c1(10, 5), c2(2, 4);
   Complex c3 = c1 + c2;
   c3.print();
}
```

Output

12 + i9

Overloading ++ as a Prefix Operator

```
class Count
{
 private:
 int value;
 public:
 Count()
{
        value=5;
}
void operator ++ ()
{
      ++value;
}
void display()
{
      cout << "Count: " << value << endl;
}};
```

```
int main()
{
    Count count1;
    ++count1;
    count1.display();
    return 0;
}
```

Output

Count: 6

Things to Remember in C++ Operator Overloading

1. By default, operators = and & are already overloaded in C++.For example, we can directly use the = operator to copy objects of the same class. Here, we do not need to create an operator function.
2. We cannot change the precedence and associativity of operators using operator overloading.
3. We cannot overload following operators in C++:

    1. :: (scope resolution) 2. . ()

    4. ?: (ternary operamember selection)

    3. .* (member selection through pointer to functiontor)

    5. sizeof operator 6. typeid Operator

4. We cannot overload operators for fundamental data types like int, float, etc

Virtual function and pure virtual function

Virtual function

- When we use the same function name in both the base and derived classes, the function in base class is declared as virtual using the keyword virtual preceding its normal declaration.
- When a function is made virtual, C++ determines which function to use at run time based on the type of object pointed to by the base pointer, rather than the type of the pointer.
- So, by making the base pointer to point to different objects, we can execute different versions of the virtual function.
- They are mainly used in Runtime polymorphism

Rules for virtual function

- The virtual function must be members of some class.
- They cannot be static members.
- They are accessed by using object pointers.
- A virtual function can be a friend of another class.
- A virtual function in a base class must be defined, even though it may not be used.
- The prototypes of the base class version of a virtual function and all the derived class versions must be identical. If two functions with the same name have different prototypes, C++ considers them as overloaded functions.
- We cannot have virtual constructors, but we can have virtual destructors.
- While a base pointer can point to any type of the derived object, the reverse is not true.
- If a virtual function is defined in the base class, it need not be necessarily redefined in the derived class. In such cases, calls will invoke the base function.

Example:

```
class base
{
        protected:
        int b;
        public:
        base(int b1)
        {
                b=b1;
        }
        virtual void display ( )
        {
        cout<<"b= "<<b;
        }
};
```

```
class derived: public base
{
        protected:
        int d;
        public:
        derived(int b1, int d1):base(b1)
        {
                d=d1;
        }
        void display ( )
        {
        cout<<"b= "<<b;
        cout<<"d= "<<d;
        }
};
```

Example:

```
void main()
{
        base b;
        base b1(5);
        b=&b1;
        b->display( );
        derived d1(10,20);
        derived d2(30,40);
        b=&d1;
        b->display( );
        b=&d2;
        b->display( );
}
```

Output:

b=5

b=10 d=20

b=30 d=40

Pure virtual function

- In the pure virtual function, the function is declared as virtual inside the base class and redefine it in the derived classes,
- The function inside the base class is used for performing any task.
- It only serves as a placeholder.
- It is also called as a “do-nothing” function.
- It may be defined as follows:

```
virtual void area( )=0;
```

- Such functions are called as pure virtual functions.
- A pure virtual function is a function declared in a base class that has no definition relative to the base class.
- A class containing pure virtual functions cannot be used to declare any objects of its own. And such classes are called as abstract base classes.

Example:

```
class shape
{
        protected:
        float r;
        public:
        shape(float r1)
        {
                r=r1;
        }
        virtual float area( )=0;
};
```

```
class circle: public shape
{
        public:
        circle(float f1):shape(r1)
        {
        }
        float area( )
        {
                return (3.14*r*r);
        }
};
```

Example:

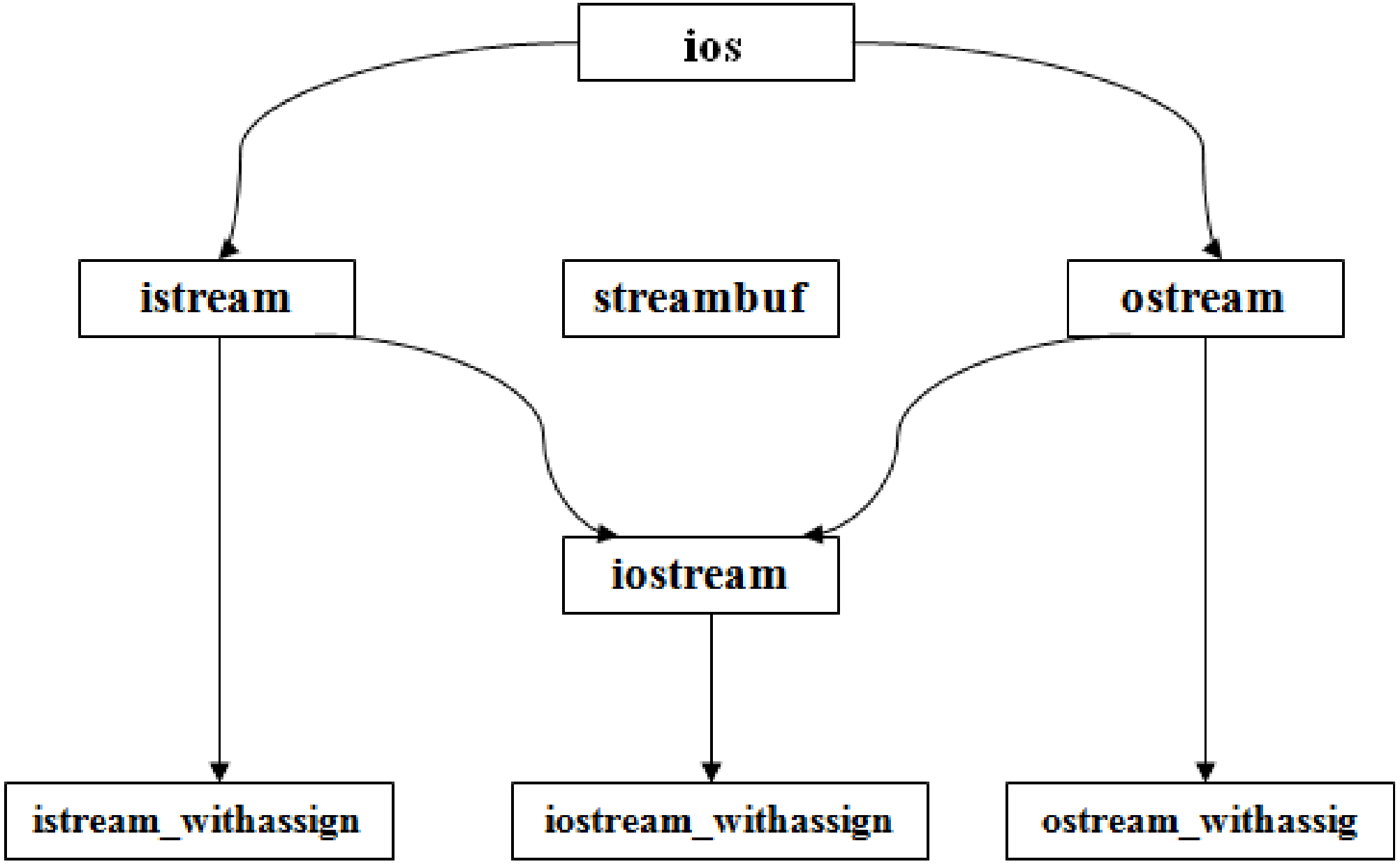
```
class rect: public shape
{
        private:
        float b;
        public:
        circle(int r1,int b1):shape(r1)
        {
                b=b1;
        }
        float area( )
        {
        return (r*b);
        }
};
```

```
void main( )
{
shape *s1,*s2;
s1=new circle(2);
s2=new rect(5,10);
s1->area( );
s2->area( );
getch( );
}
```

Output:

Introduction to File Stream Classes

- The C++ I/O system contains a hierarchy of classes that are used to define various streams to deal with both the console and disk file.
- These classes are called as stream classes.
- The stream classes are used for input and output operations with the console unit.
- These classes are declared in the header file iostream.
- The ios class is the base class for istream and ostream.
- The istream and ostream classes are base classes for iostream.
- The class ios is declared as the virtual base class so that only one copy of its members is inherited by the iostream.
- The class ios provides the basic support for formatted and unformatted I/O operations.


ios
istream
streambuf
ostream
iostream
istream_withassign
iostream_withassign
ostream_withassig

- To read and write from a file we are using the standard C++ library called fstream. Let us see the data types define in fstream library is:

Data Type	Description
fstream	It is used to create files, write information to files, and read information from files.
ifstream	It is used to read information from files.
ofstream	It is used to create files and write information to the files.

Opening and closing a file- File opening modes.

- File handling is used to store data permanently in a computer. Using file handling we can store our data in secondary memory

File Operations

1. Creating a new file. (fopen with mode “a” or “w”)
2. Opening an existing file. (fopen)
3. Reading from or writing information to the file. (fscanf or fgets, fprintf or fputs)
4. Closing the file. (fclose).

File Opening modes in C+

Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen( ) returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen( ) returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append.	Data is added to the end of the file. If the file does not exist, it will be created.
ab	Open for append in binary mode.	Data is added to the end of the file. If the file does not exist, it will be created.

Mode	Meaning of Mode	During Inexistence of file
r+	Open for both reading and writing.	If the file does not exist, fopen( ) returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen( ) returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

Working with files

- When working with the files, there is a need to declare a pointer of the type file. This file-type pointer is needed for communication between the file and the program.

Syntax:

file *file_pointer;

Example:

file *fptr;

Opening a file

- Opening a file is done using the fopen() function in the header file stdio.h.

Syntax:

file_pointer = fopen(“file_name”, “mode”);

Example:

fptr= fopen(“D:\\myfolder\\file1.txt”, “w”);

Reading from or writing information to the file

fgets( ):

- The fgets() function in C++ reads a specified maximum number of characters from the given file stream.
- On success, the fgets() function returns str and on failure it returns null pointer.

Syntax: fgets(char* str, int count ,FILE* stream);

Example: fgets(str,10, fptr);

fputs( ):

- The fputs() function in C++ writes a string completely except the terminating null character to the given output file stream.

Syntax: fputs(char* str, FILE* stream);

Example: fputs(“HELLO” , fptr);

Reading from or writing information to the file

Closing a file

- The file should be closed after reading or writing.
- Closing a file is performed using the fclose() function.

  Syntax: fclose(file_pointer);

  Example: fclose(fptr);

Example:

```
#include <fstream>
#include <iostream>
using namespace std;
int main ()
{
   char input[75];
   ofstream os;
   os.open("testout.txt");
   cout <<"Writing to a text file:" << endl;
   cout << "Please Enter your name: ";
   cin.getline(input, 100);
   os << input << endl;
  cout << "Please Enter your age: ";
   cin >> input;
   cin.ignore();
   os << input << endl;
   os.close();
```

```
   ifstream is;
   string line;
   is.open("testout.txt");
   cout << "Reading from a text file:" << endl;
   while (getline (is,line))
   {
   cout << line << endl;
   }
   is.close();
   return 0;
}
```

Example:

Output:

Writing to a text file:
 Please Enter your name: Nakul Jain
Please Enter your age: 22
 Reading from a text file: Nakul Jain
 22